Brute Force

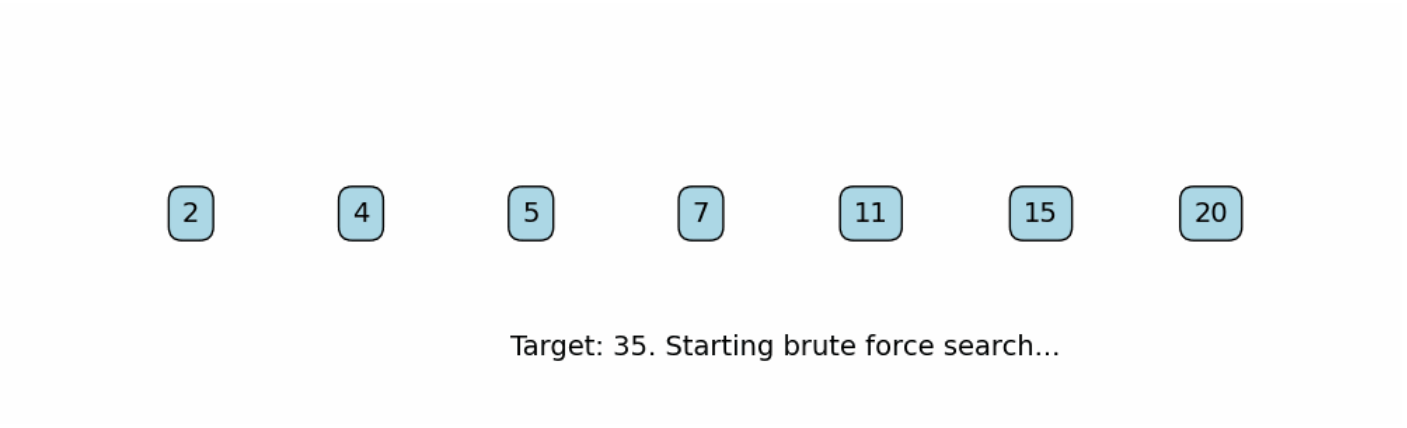
The brute force approach tries every possible combination to check for a solution, without leveraging any special properties or optimizations of the data (such as sorted order). Typically involves:

- Nested loops
  - Outer Loop traverses the array for the first element of the pair
  - Inner Loop traverses the rest of the array to find second element
- Recursion

Drawbacks:

- **Time Complexity:** Typically $O(n^2)$ or slower, impractical for large datasets.
- **Redundancy:** Many computations are repeated unnecessarily

Need to know: `Nested For Loops`, `Range`, `Len`




```
def brute_force_two_sum(nums, target):
    # Iterate through each element in the list
    for i in range(len(nums)):
        # For each element, check every other element that comes after it
        for j in range(i + 1, len(nums)):
            # Check if the current pair sums to the target
            if nums[i] + nums[j] == target:
                return (i, j)  # Return the indices as a tuple
    # If no pair is found that adds up to the target, return None
    return None

# Example usage:
```

```
nums = [2, 7, 11, 15]
target = 9

result = brute_force_two_sum(nums, target)
if result:
    print("Pair found at indices:", result)
else:
    print("No pair found that adds up to the target.")
```

Great question — **no**, nested `for` loops in Python **don't always have to use** `range()`. It totally depends on **what you're looping over**.

Let’s break this down:

When You Use `range()` in Nested Loops

You use `range()` when you're looping over **indexes**, usually for:

- Arrays
- Matrices
- Iterations with index math

```
for i in range(len(matrix)):
    for j in range(len(matrix[0])):
        print(matrix[i][j])
```

When You Don’t Need `range()`

You can also loop directly over values — nested or not — when index access isn't needed.

Example: Nested Lists

```
for row in matrix:
    for val in row:
        print(val)
```

Example: Nested Dictionaries

```
for category, items in asset_dict.items():
    for item in items:
        print(f"{category}: {item}")
```

Example: In Tech Art (Blender)

```
for obj in bpy.data.objects:
    for modifier in obj.modifiers:
        print(f"{obj.name} has modifier {modifier.name}")
```

□□ When to Use `range()` vs Direct Loop

Use Case	Use `range()`	Direct Loop (`for item in list`)
You need the index □	□ Yes	□ No
You only care about the value	□ No	□ Yes
You're modifying a list in-place	□ Often	□ Risky

So nested loops **can** use `range()`, but they **only should when** you need an index or are doing some kind of math-based traversal (e.g., comparing elements, jumping steps, etc).

Want me to show a nested loop example with `range()` from a LeetCode matrix problem or a nested asset traversal from a DCC pipeline?

Revision #6
Created 2 January 2025 22:30:38 by victor
Updated 8 April 2025 22:54:19 by victor